Generare funcții/arbori pentru evaluarea de binarizări optime

1st Radulescu Stefan
Developer and Tester
Proiect MPS
UPB, CS
stefan.rdle@gmail.com

2nd Sirbu Dan
Technical Lead and Developer
Proiect MPS
UPB, CS
dan.sirbu.sd@gmail.com

3rd Oana-Alexandra Cocoara
Project Manager
Proiect MPS
UPB, CS
alexandra.cocoara11@yahoo.com

***Abstract*—Document care contine descrierea solutiei globala. Se vor descrie rezultatele obtinute, arhitectura utilizata, cat si o sectiune care contine concluzii.**

***Index Terms*—generare arbori, python, software, analiza, solutie, implementare, agile, imagine, prag, threshold, pixel, sistem.**

I. Introducere domeniu

Proiectul are la bază conceptul de binarizare globală și locală. Scopul proiectului este acela de a combina ieșirile unor algoritmi (praguri de binarizare - numite și threshold) într-un mod inteligent pentru a obține un rezultat mult mai bun și mai general decât oricare din aceștia în mod individual.

A. Threshold

Pragul(threshold) este folosit pentru determinarea clasei unui pixel.

B. Binarizarea globală

Binarizarea globală reprezintă aplicarea unui singur prag de binarizare pe întreaga imagine și transformarea pixelilor din cadrul acesteia în valori de alb (255) sau negru (0) în funcție de valoarea pragului.

C. Obtinere prag de binarizare

Prag de binarizare se obtine pe baza unui algoritm , urmând ca pixelii care au valoarea mai mică ca respectivul prag să devină negri, iar restul albi.

II. Descriere soluție

Solutia pe care am ales-o este formata din 2 componente: parser-ele care citesc datele fisierelor(**globalParser** si **lutParser**), si **globalSolver**. Scopul Solver-ului este de a ajunge la o expresie formata din operatii matematice simple (inmultire, max, min, modul), care, aplicata pe setul de date, produce un rezultat cat mai precis. Metoda **generateCombinations** produce o lista de 14 operatori alesi complet aleatoriu, iar **generateTrees** evalueaza performanta fiecarei combinatii prin aplicarea acesteia imaginilor din fisierul GlobalTrain.csv. Rezultatul obtinut este verificat daca apartine intervalului [0, 1], iar in caz afirmativ se verifica acuratetea, consultand fisierul LuttTrain.csv, care este retinut in variabila **avg**. Media tuturor rezultatelor imaginilor reprezinta procentul de corectitudine al expresiei generate. Cel care obtine scorul maxim este retinut intr-un fisier de tip JSON, salvandu-si astfel progresul curent.

Solutia propusa pentru **localSolver**: exista un parser care citesc date din fisier (
textbfcsvParser), de unde extrag clase de pixeli(0 sau 1) si tresholdu-urile. Solutia este asemanatoare cu cea global, diferenta fiind aceea ca se verific clasa de pixeli si pe baza acestia vad daca am obtinut true positive, false positive, true negative sau flase negative, apoi le numar voi numara. Folosind numarul acesta pot calcula f-measure. In final se face o media pentru a calcula cel mai bun f-measure.

III. Arhitectură

Pentru rezolvarea proiectului am ales sa il impelmentam in Python(prin intermediul mediului de dezvoltare PyCharm), deoarece este mai usor de utilizat si ajuta la impementarea codului cat mai rapid. In rezolvarea ne-am ajutat de GitHub utilizat pentru gestionarea codului sursa cat si de servicii oferite de Google precum Google Docs(pentru usurinta in dezvoltarea decumenteleor necesare realizarii proiectului).

De asemenea, am utilizat biblioteca de parsare a fisierelor de tip json pentru a importa si exporta date, cu scopul de a retine un rezultat incremental al training-ului: ordinea operatiilor pentru cel mai bun rezultat, impreuna cu acuraterea acesteia ($\mathrm{M} \rightarrow max, m \rightarrow min, A \rightarrow modul/abs, * \rightarrow inmultire$)

Totodata, am folosit si biblioteca csv pentru parsarea fișierelor de input, impreuna cu combinations din itertools, cu care am generat toate combinatiile de operatori (max, min, abs, mul)

Proiectul a fost realizat pe urmatoarele sisteme: Specificațiile sistemelor:

A. PC 1

Hardware: CPU Ryzen 5600H, 6-core, 12-threads, 3.3GHz base, 16GB RAM software: OS Windows 11 x86-64, Python 3.10.6

B. PC 2

Hardware: CPU Ryzen 5700U, 8-core, 16-threads, 1.8GHz base, 16GB RAM software: OS Linux x86-64, Python 3.10.12

IV. Rezultate intermediare ale soluției software

Solutia Globala prezentata a ajuns la un rezultat cu o acuratete de 70.15% dupa o rulare de aproximativ 4h. Solutia Locala prezentata a ajuns la un rezultat cu o acuratete de 70.15% dupa o rulare de aproximativ 4h.

V. concluzii intermediare (incluzând abordarea globală și locală)

Algoritmul este antrenat corect, deoarece am obtinut rezultate foarte asemanatoare cu cele din fisierele de test, dupa cum se poate observa in test.json: ”tree”: ”MMm*MMm*aaa*ma”, ”initialAccuracy”: 73.46744377850419, ”factualAccuracy”: 73.46744377850419

Solutia Locala ruleaza considerabil mai mult decat cea Globala, acest lucru se datoreaza faptului ca pentru fiecare arbore trebuie sa citim toate fisierele csv.